IIKH Implementation

- The Interactive, Intelligent Kitchen Helper -

Object Oriented Programming

class 03

Team Members

곽서현, 권정주, 나선우, 오휘민, 최승훈

Speakers

곽서현, 최승훈

Contents

(a) Project Summary

❖ IIKH (Interactive Intelligent Kitchen Helper)

The IIKH system, based on object-oriented principles, leverages modern C++ features to enhance both performance and efficiency. It incorporates responsibility-driven design, where key components of the system are delegated specific tasks, promoting modularity and independence. Using features like structured bindings, constant references, and the `std::format` library, IIKH ensures streamlined data processing and minimal memory overhead. The design also emphasizes file handling with file system libraries and data parsing using regular expressions, aiming for high compatibility and maintainability across systems. Warnings are integrated to prevent overlooked return values, making the system robust and user-friendly.

One special thing in our IIKH implementation is that we used modern C++ to create the program. With powerful features of modern C++, we could achieve follow things:

- Detect whether the current system is compatible with IIKH **at compilation time** (#define-d macros and **static assertion**)
- Parse std::map<K, V> with **structured bindings.**
- Avoid copy overhead by using **constant reference** in class parameters and **emplace_back** in std::list<T>.
- Read datafiles using **filesystem libraries** to locate path, and **regular expressions** to parse data.
- Format strings with **std::format** when printing to stdout or file.
- Warns with **[[nodiscard]] attribute** if the return value of getter functions are ignored.
- Stores the filename of the recipe database and plans using **constant expressions** of non-owning string **std::string_view** (instead of using std::string), so that IIKH uses less memory on runtime.

(b) Requirements for Compiling & Executing

Our IIKH implementation is based on *modern C++*. Generally, modern C++ refers to C++11 or later. The C++ standardization committee releases a new version of C++ every 3 years (since 2011), and we are using C++20, ISO/IEC 14882:2020.

The IIKH runs on most desktop operating systems, including Windows and Linux. GNU Make and GNU Compiler Collection (a.k.a. GCC) are required to compile this program. Windows users need to manually download and install them, and Linux

users can simply install with each distribution's package manager – apt (Ubuntu), yum/dnf (Fedora), pacman (Arch), etc.

To compile IIKH in Linux, run:

```
iikh-root$ make && ./iikh
    OR
iikh-root$ g++ -o iikh *.cpp -std=c++20 && ./iikh
```

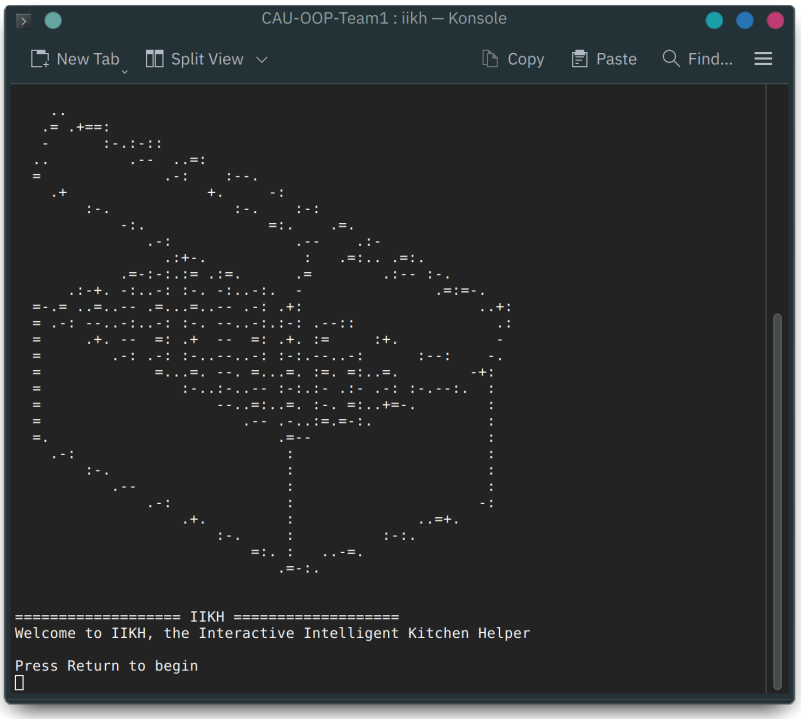
If you are compiling on Windows, replace './iikh' with '.\iikh.exe' and 'make' with 'mingw32-make', in case you can't find the 'make' command in your system.

```
iikh-root> make && ./iikh
    OR
iikh-root> g++ -o iikh.exe *.cpp -std=c++20 && .\iikh.exe
```

As IIKH uses a system command 'cls' (in Windows) or 'clear' (in macOS/Linux), please check if the command above is working on your system.

(c) Functionalities

- ❖ Greeter class



The Greeter is loaded as soon as the user runs IIKH. It shows an ASCII art picture, made from the original PDF file. The Greeter class prompts the user to select an

option from a menu displayed on the screen. 6 options are provided and waits for the user's selection. After interacting with each menu, the screen is cleared to maintain a clean and organized user interface.

The key interaction of the Greeter is evolving around a continuous loop that allows the user to navigate the main menu options. After completing each action, the user is returned to the menu to select another one. The loop continues infinitely until the user chooses the "Quit" option.

❖ RecipeDatabase class

The RecipeDatabase class manages the list of recipes. It automatically reads from the database file when its constructor is called, and saves the file while destructing. It contains the path object from std::filesystem and std::fstream to read & write file. It allows the user to search recipes(searchRecipes), add a new one(addNewRecipe), and edit a recipe(editRecipe). It has a getter function that returns a Recipe instance, and operator[] to conveniently use the getter. Also, it has a removeRecipe function that removes a recipe, which has the same name with parameter, from the database.

❖ Recipe class

The recipe constructor allows you to create a new recipe object. The recipe attributes such as name, ingredient list, instructions, and preparation time can be retrieved using getter methods. To modify the recipe, you call the edit method, and the process of updating the ingredient list, instructions, and preparation time will proceed step by step. To view the recipe, you call the displayRecipe method, which displays the recipe name, ingredient list, instructions, and preparation time in sequence.

❖ PlanManager class

The PlanManager class provides 4 key functionalities for managing and storing meal plans in a structured way. Its primary responsibilities follow:

Reading Existing Plans: The PlanManager class reads and loads all plans from a text file “plan.txt”. The plans consist of a date and a list of meals, and each meal consists of multiple recipes with serving information. A memo can be stored with each plan, but it is optional. This functionality ensures that users can retrieve previously saved plans.

Reviewing and Editing Plans: The system allows users to review all existing plans and edit them. Users can view the meal and recipe details for each plan and are given options to add or remove recipes or make changes to memos.

Creating New Plans: Users can create a new meal plan by specifying the date and adding meals with their corresponding recipes. And then the plan is saved to the plans list, which is later stored in the file when the program terminates.

Writing all Plans into File: The class writes all plans back into “plan.txt” when the program finishes. Each plan contains a date, memo, list of meals, and recipes, all stored in a predefined format.

❖ Date class

Responsible for managing memos, meal plans, and creating grocery lists related to a specific date.

1. Constructors

Date(int year, int month, int date) : Takes the year, month, and day as parameters and initializes the memo to an empty string by default.
Date(const std::string &date) : Parses the given date string and initializes the object.
Date(const std::string &date, const std::string &description) : Takes a date string and a memo as parameters and initializes the object, used to store both the date and the memo.
Date(const std::string &date, const std::string &description, const std::list<Meal> &meals) : Takes a date string, a memo, and a list of meals as parameters and initializes the object.

2. Methods

- getDate : Returns the year, month, and date as a tuple.
- getDateAsString : Returns the date as a string in the “YYYY-MM-DD” format.
- getMeals : Returns the list of meals saved in the Date object.
- getMemo : Returns the memo saved in the Date object.
- displayAndEdit : Outputs the current date and memo and provides a method to edit the memo.
- manageMeals : Allows the user to add, modify, or delete meal plans for a specific date.
- setMeals : Resets the list of meals. Allows for setting meal plans for a specific date all at once.
- buildGroceryList : Creates a list of ingredients and their required quantities for all meals on the given date. Compiles the ingredients needed from each meal's recipe to generate the grocery list.
- operator< && operator== : Overloads the comparison operators for comparing dates.

❖ Meal class

The Meal class is designed to allow users to manage various information about a single meal. A Meal object stores and manages the necessary recipe, serving size, and date information for each meal, enabling users to create individual meal plans. Users can adjust the default serving size of a recipe to the desired amount, and this adjustment will automatically update the quantity of ingredients in the recipe. The Meal class also allows the management of multiple recipes in a list, enabling users to add several recipes to a single meal plan. Additionally, the Meal class interacts with the Date class, allowing users to create meal plans tailored to specific dates, providing an easy way to organize personalized meal schedules.

(d) Implementations

❖ Greeter class

The constructor of the Greeter class takes two arguments : RecipeDatebase, and PlanManager. Both arguments are reference types. This allows the Greeter class to only access and manipulate the data related to meal plans and recipes, through the two classes. This implies that actual data handling is performed by the other manager classes.

Before the user chooses to quit, the program keeps running by the run() method, which contains an infinite loop. The printMenu method shows the user a list of options such as searching for recipes, adding a new recipe, editing existing recipes, reviewing plans, and so on. The getUserOption method prompts the user until a valid option (1 through 6) is selected. And then the method checks the input whether it is valid or not. The run method processes the user's choice by using a switch statement, and each case corresponds to some actions. Options 1 to 3 are connected to the method of RecipeDatabase class, while options 4 and 5 are connected to the PlanManager class.

The clearScreen method handles the platform-specific operation of clearing the terminal screen. It uses platform-specific system commands to clear the console, ensuring consistent behavior across different operating systems, such as Windows and Unix. To improve the efficiency, it uses **constexpr if statement** to determine which command should be executed **at compilation time**.

❖ RecipeDatabase class

When the constructor is called, it searches the data directory with std::filesystem::path, and creates if it does not exist. After then, it reads the file line by line with std::fstream, constructs a Recipe instance and adds it to the list. We are using **emplace_back** instead of traditional push_back, to avoid unnecessary copy operation. The destructor opens the file the same way, and writes data to the file. It uses **std::format** to create formatted strings.

The searchRecipes function allows users to interactively search recipes from the database. After getting input using std::getline, it uses istream_iterator to parse keywords by spaces. If multiple keywords were given, it performs “AND” operations to search the recipe. The search items are 1) recipe name, 2) instructions, and 3) ingredients. The search result is created as a list. The function, therefore, iterates the list to show the search results to the user.

getRecipe function and operator[] are designed to return the Recipe instance from the internal list. The difference between them is that getRecipe returns an empty Recipe object if the given name is not found, while [ ] first pushes a Recipe object into the database and returns its reference.

The addNewRecipe function also interactively adds a new recipe to the database. If the same name already exists, it returns. Otherwise, it creates a Recipe instance with the given name and yields the control to the Recipe::edit.

editRecipe allows the user to edit a recipe in the database. If the recipe is not found, it returns. If the recipe exists, the user can edit or remove the recipe. Editing recipe delegates the editing job to Recipe class, and removing recipe simply removes it with remove_if method.

In this class, cin.ignore is used to clear the input buffer and cin.get to pause the program. (getting a ‘Return’ key input)

❖ Recipe class

First, the recipe constructor creates a new recipe by taking the recipe name, ingredients (names and quantities), cooking instructions, and preparation time as arguments. When creating a new recipe, the data types are specified as follows: the name is a string, the ingredients use a map class to match the ingredient names and their quantities, the instructions are a string, and the preparation time is an integer. The getter methods use the **`[[nodiscard]]`** attribute to ensure that the return values are not ignored. The getIngredients method returns the ingredients object, but the other get methods return a single value.

In the edit method, the first step is to clear the values stored in ingredients. Then, the user is prompted to input new ingredients. The input format is *<ingredient name> <ingredient quantity> <ingredient name> <ingredient quantity> …* in a continuous form. Before processing the input, std::cin.ignore is used to remove any newline characters that might remain from previous input. After that, the input entered by the user is read as a single line and stored in the line variable. To split the string in line by spaces, an `istringstream` object (iss) is created, which allows treating the string as an input stream. The ingredient name and quantity are read sequentially from the input stream, with the name stored as the map key and the quantity as the value. Once the ingredient list is updated, the cooking instructions are modified by receiving a new string via getline, and the preparation time is updated by inputting an integer value representing the minutes. Finally, std::cin.ignore() is used to clear the input buffer.

The displayRecipe method prints the recipe name, ingredient list, instructions, and preparation time in order using std::cout. The ingredient list is iterated through the map object, and each pair of name and quantity is printed to display all the ingredients.

❖ PlanManager class

The implementation of the PlanManager class has several implementation issues.

File Handling: The file “plans.txt” must store a lot of information about the set of plans, so it is necessary to manage efficiently. So, the PlanManager class utilizes the C++ <fstream> and <regex> library to read and write the meal plans to a file (plan.txt) as a structured format. The file contains structured data in the following format:

```
YYYY-MM-DD $#memo#$ [Meal]={Recipe,Recipe,...,Servings}
```

The system reads and parses this file using **regular expressions** (<regex>) to match dates, meals, recipes, and servings. The regex library ensures that the data is read correctly and stored in appropriate objects.

The file reading functionality is automatically executed when an object of the class PlanManager is created, through the constructor of C++. The file writing functionality is automatically executed when an object of the class is destroyed, through the destructor of C++, i.e., when the program terminates.

Data Parsing and Storage: The plans are stored in a std::list container, where each entry consists of a Date object and a list of Meal objects. This allows for easy manipulation of the plans, like sorting, modifying, and searching.

Plan Review and Editing: To display the plans, the system uses iterators to traverse the plans list and call some functions from other classes. Date::displayAndEdit() function is called to show the date information and edit the memo. Similarly, the displayMealInfo() function is used to show details about each meal. If a user needs to modify the information of a meal, functions like addRecipe() and removeRecipe() are also invoked.

Creating New Plans: First, the system prompts the user to input a new date and checks whether a plan already exists for that date using the ranges::find() function. If not, the user can enter meal information, which is stored in a Meal object. The new plan is then added to the plans list, and the list is sorted using the function sortPlans() to ensure the plans to be stored in consistent order.

Sorting Plans: The plans are stored in a sorted manner based on the date. This is achieved by using the sort() function, which compares two Date objects. This ensures that when plans are displayed or written to a file, they appear in the correct chronological order.

❖ Date class

1. Constructor Implementation

The constructor Date(const std::string &date, const std::string &description, const std::list<Meal> &meals) expands upon previously defined constructors by adding a list of meals (Meal list). It initializes the date and memo by calling the earlier constructors, then sets the meals list for the specified date. By reusing existing constructors, it creates a multi-constructor.

2. Meal Management Approach [manageMeals()]

This method provides functionality for managing meals for a specific date. It implements an interface that allows adding or deleting meals, taking user input to perform tasks. When adding a new meal, a `Meal` object is created, and recipes are added to it. The input is received via `cin`, and the method checks whether the meal name already exists. If it does not exist, a new meal is added. This method is designed to be simple and intuitive, allowing the list to be dynamically modified through an imperative interface.

3. Grocery List Generation Approach [buildGroceryList()]

This method compiles all the ingredients needed from the meals for a specific date into a grocery list. It calls getGroceryList() from the Meal object to retrieve the ingredients and their quantities for each meal, then stores them in a map called groceryList, where the ingredient names are the keys and the quantities are the values. If duplicate ingredients are present, their quantities are summed.

4. Operator Overloading Approach

The ‘<’ operator is overloaded to compare dates. It is implemented to compare the year, month, and day in sequence, returning true if the year is smaller, if the month is smaller in the same year, or if the day is smaller in the same month and year.

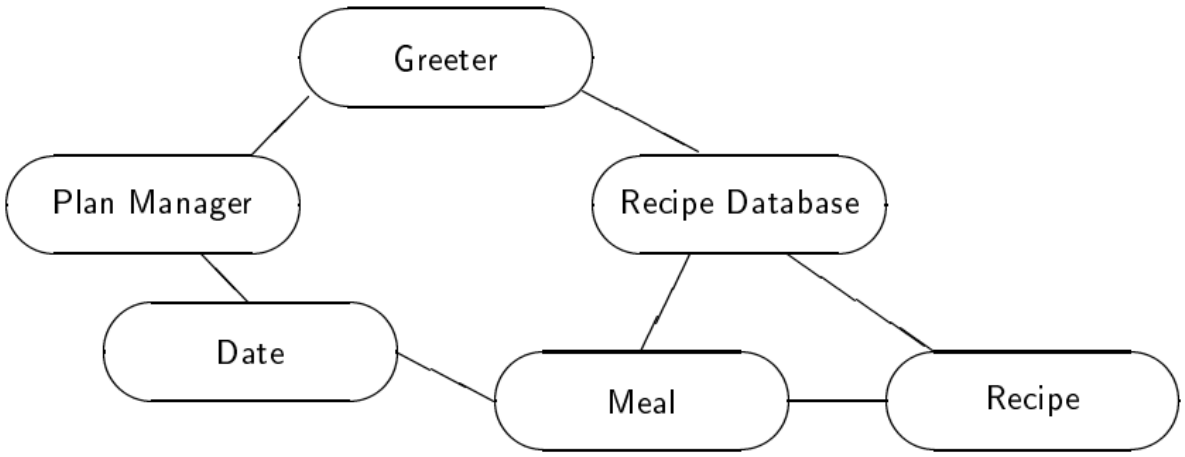
5. Memo Management

The displayAndEdit() method displays the current date and memo, allowing the user to modify the memo. It is implemented as a simple console-based interface, where the user can modify the memo and save it after displaying it.

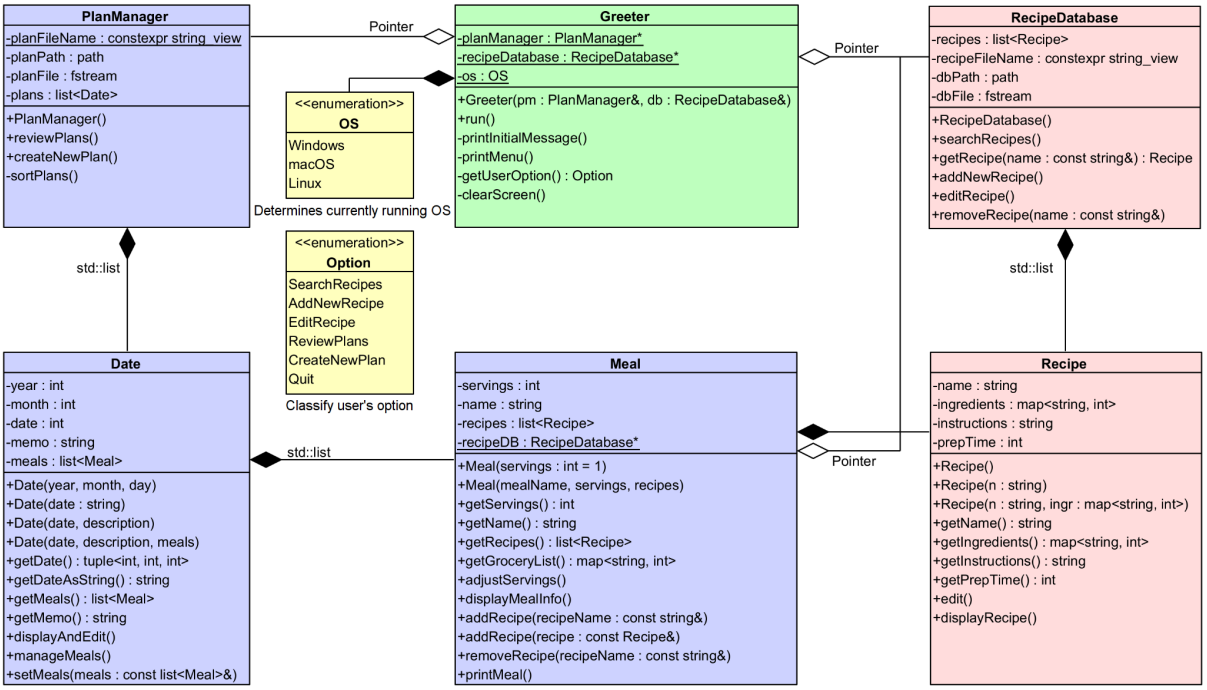
❖ Meal class

One of the most important features implemented in the Meal class is the serving size adjustment functionality. The adjustServings() method was designed to automatically adjust the quantity of ingredients in a recipe based on the user's input for the number of servings. This was implemented by comparing the default serving size of the recipe with the input value, and proportionally changing the ingredient quantities. Additionally, the Meal class provides functionality to manage multiple recipes in a list, and the addRecipe() method allows new recipes to be added to this list. A key issue during implementation was ensuring that each recipe was correctly assigned to the list and managed accurately. The Meal class also interacts with the Date class to manage meal plans by date, collaborating with the PlanManager class to ensure that Meal objects are properly associated with the correct dates and managed efficiently. The primary challenge here was ensuring that meal plans were saved and managed without any data being omitted or mismanaged across different dates.

(e) Program Design



< A component diagram in the given PDF file >



< Final class diagram of IIKH >

The picture above is the class diagram of IIKH. You can see the relationships among classes, as well as the structure of each class. The black diamond in the line between classes stands for a composition, where one class is a part-of other class. For example, as RecipeDatabase stores the list of Recipe, they are in a composition relationship. The white diamond, on the other hand, refers to an aggregation. Neither is one class a parent of the other class, nor has the other inside itself. Using a pointer, one class can refer to the other class.

(f) Execution Results

<Search recipes> : select option 1, and type keywords (a recipe name) to get an information of the recipe, including ingredients, instructions, preparation time.

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 1
Input keywords (separated by spaces): 초밥

Search results:

Recipe Name: 광어초밥
  Ingredients:
    광어 : 100g
  Instructions: 초밥용 쌀을 식초로 간을 하고 광어를 얹어 초밥을 만든다.
  Preparation Time: 10 minutes
Recipe Name: 사케동
  Ingredients:
    연어 : 100g
  Instructions: 초밥을 준비한 후 연어를 얹고 간장을 뿌린다.
  Preparation Time: 10 minutes
Recipe Name: 연어초밥
  Ingredients:
    연어 : 100g
  Instructions: 초밥용 쌀을 식초로 간하여 연어를 얹어 초밥을 만든다.
  Preparation Time: 10 minutes
Recipe Name: 장어초밥
  Ingredients:
    장어 : 100g
  Instructions: 초밥용 쌀에 장어를 얹고 간장을 뿌려 초밥을 만든다.
  Preparation Time: 10 minutes
Recipe Name: 참치초밥
  Ingredients:
```

As described in the given PDF file, it uses the ‘AND’ search mechanism. If the user types multiple keywords, it only displays the recipes that all of the keywords are included.

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 1
Input keywords (separated by spaces): 초밥 광어

Search results:

Recipe Name: 광어초밥
  Ingredients:
    광어 : 100g
  Instructions: 초밥용 쌀을 식초로 간을 하고 광어를 얹어 초밥을 만든다.
  Preparation Time: 10 minutes

Press Return to continue...
```

<Add a new recipe in database>

Added new recipe ‘Noodles’ via Menu 2

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 2
Enter the new name of the recipe: 잔치국수
Enter ingredients (format: egg 100 flour 200 ...): 소면 150 당근 50 양파 50 김치 50
멸치 30 다시마 10
Enter instruction: 소면을 삶은 후, 멸치와 다시마로 우려낸 국물을 끼얹고 삶은 채소와
김치를 얹는다.
Enter preparation time (minutes): 30

Recipe added successfully.
Press Return to continue...
```

After adding the new recipe

```
Input keywords (separated by spaces): 국수

Search results:

Recipe Name: 잔치국수
  Ingredients:
    김치: 50g
    다시마: 10g
    당근: 50g
    멸치: 30g
    소면: 150g
    양파: 50g
  Instructions: 소면을 삶은 후, 멸치와 다시마로 우려낸 국물을 끼얹고 삶은 채소와 김치
를 얹는다.
  Preparation Time: 30 minutes

Press Return to continue...
```

If already exists, shows an error

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 2
Enter the new name of the recipe: 잔치국수

Recipe already exists. Aborting!
Press Return to continue...
```

<Edit recipes>

Edit recipe with ‘Edit a recipe’ > ‘Edit’ menu

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 3
Enter the name of the recipe to edit: 잔치국수
Edit or remove the recipe '잔치국수'

  1. Edit
  2. Remove
  3. Cancel

Enter your choice > 1
Enter ingredients (format: egg 100 flour 200 ...): 잔치국수키트 500 물 400
Enter instruction: 키트에 물을 넣고 끓인다.
Enter preparation time (minutes): 10
Recipe edited successfully.
Press Return to continue...
```

And check if the recipe is updated.

```
Input > 1
Input keywords (separated by spaces): 국수

Search results:

Recipe Name: 잔치국수
  Ingredients:
    물: 400g
    잔치국수키트: 500g
  Instructions: 키트에 물을 넣고 끓인다.
  Preparation Time: 10 minutes

Press Return to continue...
```

Or you can **remove** the recipe.

```
Input > 3
Enter the name of the recipe to edit: 잔치국수
Edit or remove the recipe '잔치국수'

  1. Edit
  2. Remove
  3. Cancel

Enter your choice > 2
Recipe removed successfully.
Press Return to continue...
```

After that, you cannot search for it.

```
Input > 1
Input keywords (separated by spaces): 국수

No recipes found.

Press Return to continue...
```

<Review all plans>

This allows the user to iterate the plans and check the grocery list.

```
Input > 4

====== 2024-10-03 (개천절) ======
Would you like to edit the memo? (Y/N): N
No changes made to the memo.

Meals: 파스타, 초밥세트

Information about 파스타 (2 servings)
Recipes included in this meal :
 - 까르보나라 (for 2 servings)
Recipe Name: 까르보나라
  Ingredients:
    계란: 60g
    베이컨: 50g
    스파게티: 100g
    파마산치즈: 30g
  Instructions: 스파게티를 삶고 베이컨과 계란을 섞어 소스를 만든 후 버무린다.
  Preparation Time: 15 minutes
 - 마르게리타피자 (for 2 servings)
Recipe Name: 마르게리타피자
  Ingredients:
    모짜렐라치즈: 100g
    토마토소스: 50g
    피자도우: 400g
  Instructions: 도우에 토마토소스와 치즈를 올리고 220도에서 10분간 구운다.
  Preparation Time: 15 minutes
Grocery list for this date:
=> 계란 (120g)
=> 모짜렐라치즈 (200g)
=> 베이컨 (100g)
=> 스파게티 (200g)
=> 토마토소스 (100g)
=> 파마산치즈 (60g)
=> 피자도우 (800g)
```

- Optionally, the user can edit the memo of a day.

```
====== 2024-10-03 (개천절) ======
Would you like to edit the memo? (Y/N): Y
Enter new memo: 고조선 건국 기념일
New Memo: 고조선 건국 기념일
Memo updated successfully.
```

- If so, the memo changes when you run the menu 4 again.

```
Input > 4

====== 2024-10-03 (고조선 건국 기념일) ======
Would you like to edit the memo? (Y/N): N
No changes made to the memo.
```

<Add a recipe to meal>

```
=> 토마토소스 (100g)
=> 파마산치즈 (60g)
=> 피자도우 (800g)
If you want to edit meal, type either 1 or 2:

  1. Add a recipe to meal
  2. Remove a recipe from meal
  3. Cancel (Go to next meal/date)
  4. Return to main menu

Input > 1
Enter the name of the recipe to add: 사케동
```

After adding a recipe to a meal, it shows after you review the plan once again.

```
Recipe Name: 사케동
  Ingredients:
    연어 : 100g
  Instructions: 초밥을 준비한 후 연어를 얹고 간장을 뿌린다.
  Preparation Time: 10 minutes
Grocery list for this date:
=> 계란 (120g)
=> 모짜렐라치즈 (200g)
=> 베이컨 (100g)
=> 스파게티 (200g)
=> 연어 (200g)
=> 토마토소스 (100g)
=> 파마산치즈 (60g)
=> 피자도우 (800g)
If you want to edit meal, type either 1 or 2:
```

<Remove a recipe from meal>

```
If you want to edit meal, type either 1 or 2:

  1. Add a recipe to meal
  2. Remove a recipe from meal
  3. Cancel (Go to next meal/date)
  4. Return to main menu

Input > 2
Enter the name of the recipe to remove: 사케동
```

If you remove a recipe from a meal,

```
Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 4

====== 2024-10-03 (개천절) ======
Would you like to edit the memo? (Y/N): N
No changes made to the memo.

Meals: 파스타, 초밥세트

Information about 파스타 (2 servings)
Recipes included in this meal :
 - 까르보나라 (for 2 servings)
Recipe Name: 까르보나라
  Ingredients:
    계란 : 60g
    베이컨 : 50g
    스파게티 : 100g
    파마산치즈 : 30g
  Instructions: 스파게티를 삶고 베이컨과 계란을 섞어 소스를 만든 후 버무린다.
  Preparation Time: 15 minutes
 - 마르게리타피자 (for 2 servings)
Recipe Name: 마르게리타피자
  Ingredients:
    모짜렐라치즈 : 100g
    토마토소스 : 50g
    피자도우 : 400g
  Instructions: 도우에 토마토소스와 치즈를 올리고 220도에서 10분간 구운다.
  Preparation Time: 15 minutes
```

The recipe disappears from the meal list.

<Create a new plan (memo, meal name, list of recipes, serving)>

Created a new meal plan and added a recipe to the meal.

```
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > 5
Enter the date to make your plan (YYYY-MM-DD) : 2024-10-01

====== 2024-10-01 (No memo) ======
Would you like to edit the memo? (Y/N): Y
Enter new memo: 국군의날
New Memo: 국군의날
Memo updated successfully.

Meals for 2024-10-01
No meals planned for this date.

What would you like to do? Choose one.

  1. Add a meal
  2. Remove a meal
  3. Exit

Enter your choice number: 1
Enter the name of the meal to add: 중식
Enter the number of servings: 2
Add recipes to the meal, separated by space >> 짜장면
Meal added successfully.
Meals for 2024-10-01

What would you like to do? Choose one.

  1. Add a meal
  2. Remove a meal
  3. Exit

Enter your choice number: 3
```

As a result, we could search the plan in the program.

```
====== 2024-10-01 (국군의날) ======
Would you like to edit the memo? (Y/N): N
No changes made to the memo.

Meals: 중식

Information about 중식 (2 servings)
Recipes included in this meal :
 - 짜장면 (for 2 servings)
Recipe Name: 짜장면
  Ingredients:
    돼지고기: 50g
    면: 100g
    짜장소스: 50g
  Instructions: 면을 삶고 짜장소스를 부어 볶아낸다.
  Preparation Time: 15 minutes
Grocery list for this date:
=> 돼지고기 (100g)
=> 면 (200g)
=> 짜장소스 (100g)
If you want to edit meal, type either 1 or 2:

  1. Add a recipe to meal
  2. Remove a recipe from meal
  3. Cancel (Go to next meal/date)
  4. Return to main menu

Input >
```

(g) OOP in IIKH & Our Reviews

In this IIKH project, we applied several object-oriented programming concepts to improve the development and structure of the system. We focused on breaking down the project into key classes such as Meal, Recipe, RecipeDatabase, Date, PlanManager, and Greeter. Each class was designed to encapsulate its data and behavior, following the principle of modularity.
We applied composition in the ‘Date’ class to hold a list of ‘Meal’ objects, allowing each ‘Date’ instance to manage its own set of meals. This setup made it easier to manage meal plans over time, which the ‘PlanManager’ class handled by mapping dates to meals. The ‘PlanManager’ also leveraged this composition structure to ensure meal plans were well-organized.

Additionally, we utilized static variables in the ‘Meal’ class to share a common ‘RecipeDatabase’ across all meal instances. This use of static variables ensured consistency in recipe data across all meals and improved memory efficiency by preventing the duplication of recipe information.

While working on the project, we also implemented the Greeter class, which was responsible for user interaction. This class encapsulated the logic for greeting users based on specific conditions, providing a personalized experience when users accessed the system. The separation of this functionality into its own class kept the user interface code clean and modular.

In this project, polymorphism was applied in the way ‘PlanManager’ interacted with different types of ‘Meal’ objects. This allowed the system to handle different meal types using a common interface, ensuring flexibility in future system enhancements without requiring significant changes to the existing structure.

Through the project, we can come to appreciate the value of encapsulation and abstraction in designing robust and maintainable systems. By keeping the internal workings of each class hidden and only exposing the necessary functionality, we ensured that the system was easier to update and maintain. We also learned how well-planned object relationships can prevent tight coupling, making the system more flexible and adaptable to changes.

During this team project, it also helps me become familiar with git for collaboration. Using git, we learned to track code changes, create branches for new features, and manage pull requests to integrate team members' work. This experience reinforced the importance of version control in maintaining a stable codebase and resolving conflicts efficiently.

Overall, the IIKH project solidified my understanding of key OOP principles such as encapsulation, composition, and polymorphism. It also highlighted the critical role of git in managing team projects effectively. This experience taught me to approach system design with both technical soundness and collaboration in mind, preparing me to handle more complex development projects in the future.

(h) Conclusions

Through this project, we successfully implemented the IIKH: Interactive Intelligent Kitchen Helper system and efficiently completed the team project using object-oriented design. By utilizing modern C++ features such as std::format, std::filesystem, and std::regex, we facilitated data processing and file management, and also designed an intuitive interface that allows users to easily navigate the system's menu, search or edit recipes, and create and manage meal plans. By applying the principles of object-oriented programming, we distributed responsibilities and clearly defined the roles of each class, resulting in a highly maintainable and scalable program.